```
// James Le - CS 271
// set.h - A Set ADT
// Jan 25, 2017
// This implementation uses a linked list

#ifndef SET_H
#define SET_H

#include <iostream>

using namespace std;

template <class Element>
class Node
{
  public:

   Element value;
   Node<Element> *next;

   Node(Element item)
   {
      value = item;
      next = NULL;
   }
};

template <class Element>
class Set;

template <class Element>
ostream& operator<<(ostream& stream, const Set<Element>& s);

template <class Element>
class Set
{
 public:

   Set();                                   // default constructor
   Set(const Set<Element>& s);              // copy constructor
   ~Set();                                  // destructor

   void insert(const Element& x);           // add x to the set
   void remove(const Element& x);           // remove x from the set
   int cardinality() const;                 // returns size of the set
   void print();                            // helper method
   bool empty() const;                      // returns true if empty, false o/w
   bool contains(const Element& x) const;   // true if x is in set, false o/w

   bool operator==(const Set<Element>& s) const;           // equality operator
   bool operator<=(const Set<Element>& s) const;           // subset operator
   Set<Element>& operator+(const Set<Element>& s) const;   // union operator
   Set<Element>& operator&(const Set<Element>& s) const;   // intersection operator
   Set<Element>& operator-(const Set<Element>& s) const;   // difference operator

   Set<Element>& operator=(const Set<Element>& s);         // assignment operator

   // stream insertion operator
   friend ostream& operator<< <Element>(ostream& stream, const Set<Element>& s);

 private:

   Node<Element> *head;
   int length;
   Node<Element> *pointerToNode(int index);

   void copy(const Set<Element>& s);   // copy the set s to this set
   void destroy();                     // delete all elements in the set
};
```

```
#include "set.cpp"

#endif
```

```
// James Le - CS 271
// set.cpp
// Jan 25, 2017

#include <iostream>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <string>
#include <cassert>
#include <vector>

using namespace std;

template <class Element>
Set<Element>::Set() // Default constructor
{
  head = NULL;
  length = 0;
}

template <class Element>
Set<Element>::Set(const Set<Element>& s) // Copy constructor
{
  copy(s);
}

template <class Element>
Set<Element>::˜Set() // Destructor
{
  destroy();
}

// This method adds an non existence item into the set, do nothing if the item is in the
set
template <class Element>
void Set<Element>::insert(const Element& x) // Insert Method
{
        for(Node<Element> *current = head; current != NULL; current = current->next)
  {
    if(current->value == x)
    {
      cerr << "Error: Item Already in Set" << endl;
      return;
    }
  }
  Node<Element> *node = new Node<Element>(x);
  Node<Element> *newNode = new Node<Element>(x);
  if(head != NULL)
  {
    Node<Element> *curr = head;
    while(curr->next != NULL)
    {
      curr = curr->next;
    }
    curr->next = newNode;
  }
  else
  {
    head = newNode;
  }
  length++;
}

// This method removes an item in the set, do nothing if the item does not exist
template <class Element>
void Set<Element>::remove(const Element& x) // Remove Method
{
  Node<Element> *prev = NULL;
  Node<Element> *del = NULL;
  if(head->value == x)
  {
    del = head;
    head = del->next;
    delete del;
    length--;
    return;
  }
  prev = head;
  del = head->next;
  while(del != NULL)
  {
    if(del->value == x)
    {
      prev->next = del->next;
    }
    delete del;
    length--;
    break;
  }
  prev = del;
  del = del->next;
}

// This method returns the size of the set
template <class Element>
int Set<Element>::cardinality() const // Cardinality Method
{
  return length;
}

// This method returns true if the set is empty, false otherwise
template <class Element>
bool Set<Element>::empty() const // Empty Method
{
  return(this->head == NULL);
}

// This method returns true if x is in the set, false otherwise
template <class Element>
bool Set<Element>::contains(const Element& x) const // Contains Method
{
  for(Node<Element> *current = head; current != NULL; current = current->next)
  {
    if(current->value == x)
    {
      return true;
    }
  }
}

// This method returns true if the 2 sets have the same elements, false otherwise
template <class Element>
bool Set<Element>::operator==(const Set<Element>& s) const // Equality Operator
{
  Node<Element> *hNode = head;
  Node<Element> *sNode = s.head;
  int counter = 0;

  while(hNode != NULL)
  {
    while(sNode != NULL)
    {
      if(hNode->value == sNode->value)
      {
        counter++;
        break;
      }
      else
      {
        sNode = sNode->next;
      }
    }
    hNode = hNode->next;
    sNode = s.head;
  }

  if(counter == length)
  {
    return true;
  }
  else
  {
    return false;
  }
}

// This method returns true if one set is a subset of another, false otherwise
template <class Element>
bool Set<Element>::operator<=(const Set<Element>& s) const // Subset Operator
{
  Node<Element> *hNode = head;
  Node<Element> *sNode = s.head;
  int counter = 0;

  while(hNode != NULL)
  {
    while(sNode != NULL)
    {
      if(hNode->value == sNode->value)
      {
        counter++;
        break;
      }
      else
      {
        sNode = sNode->next;
      }
    }
    hNode = hNode->next;
    sNode = s.head;
  }

  if(counter == length)
  {
    return true;
  }
  else
  {
    return false;
  }
}

// This method merges 2 sets into 1, no duplicate
template <class Element>
Set<Element>& Set<Element>::operator+(const Set<Element>& s) const // Union Operator
{
  Set<Element> *result = new Set<Element>(*this);
  Node<Element> *sNode = s.head;
  Node<Element> *hNode = result->head;

  for(sNode; sNode != NULL; sNode = sNode->next)
  {
    for(hNode; hNode != NULL; hNode = hNode->next)
    {
      if(sNode->value == hNode->value)
      {
        break;
      }
    }
    result->insert(sNode->value);
  }
  return *result;
}

// This method takes all the common items from 2 sets and put into 1
template <class Element>
Set<Element>& Set<Element>::operator&(const Set<Element>& s) const // Intersection Operat
or
{
  Set<Element> *result = new Set<Element>();
  Node<Element> *sNode = s.head;
  Node<Element> *hNode = this->head;

  while(hNode != NULL)
  {
    bool unique = true;
    while(sNode != NULL)
    {
      if(hNode->value == sNode->value)
      {
        unique = false;
      }
      sNode = sNode->next;
    }
    if(unique == false)
    {
      result->insert(hNode->value);
    }
    hNode = hNode->next;
    sNode = s.head;
  }
  return *result;
}

// This method takes the items from 2 sets that are different from each other and put int
o 1
template <class Element>
Set<Element>& Set<Element>::operator-(const Set<Element>& s) const // Difference Operator
{
  Set<Element> *result = new Set<Element>();
  Node<Element> *sNode = s.head;
  Node<Element> *hNode = this->head;

  while(hNode != NULL)
  {
    bool unique = true;
    while(sNode != NULL)
    {
      if(hNode->value == sNode->value)
      {
        unique = false;
      }
      sNode = sNode->next;
    }
    if(unique == true)
    {
      result->insert(hNode->value);
    }
    hNode = hNode->next;
    sNode = s.head;
  }

  while(sNode != NULL)
  {
    bool unique = true;
    while(hNode != NULL)
    {
      if(sNode->value == hNode->value)
      {
        unique = false;
      }
      hNode = hNode->next;
    }
    if(unique == true)
    {
      result->insert(sNode->value);
    }
    sNode = sNode->next;
    hNode = this->head;
  }

  return *result;
}

template <class Element>
Set<Element>& Set<Element>::operator=(const Set<Element>& s) // Assignment Operator
{
  if(this != &s)
  {
    copy(s);
  }
  return *this;
}

template <class Element>
void Set<Element>::copy(const Set<Element>& s) // copy the set s to this set
{
  if(s.head != NULL)
  {
    head = new Node<Element>(s.head->value);

    Node<Element> *newNode = head;
    for(Node<Element> *ps(s.head->next); ps != NULL; ps = ps->next)
    {
      newNode->next = new Node<Element>(ps->value);
      assert(newNode->next != NULL);
      newNode = newNode->next;
    }
  }
  else
  {
    head = NULL;
  }
  length = s.length;
}

template <class Element>
void Set<Element>::print() // helper method
{
  Node<Element> *newNode = head;
  cout << "{";
  while(newNode != NULL)
  {
    cout << newNode->value << ", ";
    newNode = newNode->next;
  }
  cout << "Length = " << length << "}" << endl;
}

template <class Element>
void Set<Element>::destroy() // delete all elements in the set
{
  while(head != NULL)
  {
    Node<Element> *temp(head);
    head = head->next;
    delete temp;
  }
}
```

```
template <class Element>
Node<Element> * Set<Element>::pointerToNode(int index)
{
  Node<Element> *testNode = head;
  for(int j = 0; j < index; j++)
  {
    testNode = testNode->next;
  }
  return testNode;
}

template <class Element>
ostream& operator<<(ostream& stream, const Set<Element>& s) // overloading the << operato
r
{
  stream << s.print();
  s.print();
  return stream;
}
```

```
// James Le - CS 271
// test_set.cpp
// Jan 25, 2017

#include <cmath>
#include "time.h"
#include <string>
#include <ctime>
#include "set.h"
#include <cassert>
#include <iostream>

using namespace std;

void test_insert()
{
        Set<double> star;
        star.print();
        star.insert(100.00);
        star.print();
        star.insert(25.00);
        star.print();
        star.insert(50.00);
        star.print();
        star.insert(75.00);
        star.print();
}

void test_remove()
{
        Set<double> compsci;
        compsci.print();
        compsci.insert(100.00);
        compsci.print();
        compsci.insert(25.00);
        compsci.print();
        compsci.insert(50.00);
        compsci.print();
        compsci.insert(75.00);
        compsci.print();
        compsci.remove(100.00);
        compsci.print();
}

void test_cardinality()
{
        Set<double> denison;
        denison.print();
        denison.insert(100.00);
        denison.print();
        denison.insert(25.00);
        denison.print();
        denison.insert(50.00);
        denison.print();
        denison.insert(75.00);
        denison.print();
        cout << denison.cardinality() << endl;
}

void test_empty()
{
        Set<double> granville;
        granville.print();
        granville.insert(100.00);
        granville.print();
        granville.insert(25.00);
        granville.print();
        granville.insert(50.00);
        granville.print();
        granville.insert(75.00);
        granville.print();
        cout << granville.empty() << endl;
}

void test_contains()
{
        Set<double> ohio;
        ohio.print();
        ohio.insert(100.00);
        ohio.print();
        ohio.insert(25.00);
        ohio.print();
        ohio.insert(50.00);
        ohio.print();
        ohio.insert(75.00);
        ohio.print();
        cout << ohio.contains(25.00) << endl;
}

void test_equality()
{
        Set<double> havill;
        Set<double> bressoud;
        havill.insert(100.00);
        havill.print();
        havill.insert(25.00);
        havill.print();
        bressoud.insert(100.00);
        bressoud.print();
        bressoud.insert(25.00);
        bressoud.print();
        assert(havill == bressoud);
}

void test_equals()
{
        Set<double> olin;
        Set<double> talbot;
        olin.insert(100.00);
        olin.print();
        olin.insert(25.00);
        olin.print();
        talbot.insert(120.00);
        talbot.print();
        talbot.insert(5.00);
        talbot.print();
        olin.print();
        olin = talbot;
        olin.print();
}

void test_plus()
{
        Set<double> fellows;
        Set<double> knapp;
        Set<double> higley;
        fellows.insert(100.00);
        fellows.print();
        fellows.insert(25.00);
        fellows.print();
        knapp.insert(120.00);
        knapp.print();
        knapp.insert(5.00);
        knapp.print();
        higley = fellows + knapp;
        higley.print();
}

void test_subtract()
{
        Set<double> fellows;
        Set<double> knapp;
        Set<double> higley;
        fellows.insert(100.00);
        fellows.print();
        fellows.insert(25.00);
        fellows.print();
        knapp.insert(120.00);
        knapp.print();
        knapp.insert(5.00);
        knapp.print();
        higley = fellows - knapp;
        higley.print();
}

void test_intersection()
{
        Set<double> fellows;
        Set<double> knapp;
        Set<double> higley;
        fellows.insert(100.00);
        fellows.print();
        fellows.insert(25.00);
        fellows.print();
        knapp.insert(120.00);
        knapp.print();
        knapp.insert(5.00);
        knapp.print();
        higley = fellows & knapp;
        higley.print();
}

void test_subset()
{
        Set<double> fellows;
        Set<double> knapp;
        fellows.insert(100.00);
        fellows.print();
        fellows.insert(25.00);
        fellows.print();
        knapp.insert(120.00);
        knapp.print();
        knapp.insert(5.00);
        knapp.print();
        assert(fellows <= knapp);
}

void test_destroy()
{
        Set<double> slayter;
        slayter.print();
        slayter.insert(100.00);
        slayter.print();
        slayter.insert(25.00);
        slayter.print();
        slayter.insert(50.00);
        slayter.print();
        slayter.insert(75.00);
        slayter.print();
        slayter.~Set();
        slayter.print();
}

int main()
{
        test_insert();
        test_remove();
        test_cardinality();
        test_empty();
        test_contains();
        test_equality();
        test_destroy();
        test_equals();
        test_plus();
        test_intersection();
        test_subtract();
        test_subset();

        return 0;
}
```

```
// James Le - CS 271
// set.cpp
// Jan 25, 2017

#include <string>
#include <sstream>
#include <iostream>
#include <fstream>
#include "set.h"

using namespace std;

Set<string> VA;
Set<string> NY;
Set<string> MA;
Set<string> OH;
Set<string> OtherState;
Set<string> Episcopalian;
Set<string> Presbyterian;
Set<string> Methodist;
Set<string> OtherReligion;
Set<string> Forties;
Set<string> Fifties;
Set<string> Sixties;
Set<string> Whig;
Set<string> Democrat;
Set<string> Republican;
Set<string> OtherParty;

void presidentNames()
{
  string line;
  int start = -1;

  ifstream input("pres.txt");
  while(getline(input, line))
  {
    string name, party, state, religion, age;
    name = line.substr(0, line.find(’\t’, 0));
    line = line.substr(line.find(’\t’, 0) + 1, 100);

    party = line.substr(0, line.find(’\t’, 0));
    line = line.substr(line.find(’\t’, 0) + 1, 100);

    state = line.substr(0, line.find(’\t’, 0));
    line = line.substr(line.find(’\t’, 0) + 1, 100);

    religion = line.substr(0, line.find(’\t’, 0));
    line = line.substr(line.find(’\t’, 0) + 1, 100);
    age = line;

    if(state == "VA")
    {
      VA.insert(name);
    }
    else if(state == "NY")
    {
      NY.insert(name);
    }
    else if(state == "MA")
    {
      MA.insert(name);
    }
    else if(state == "OH")
    {
      OH.insert(name);
    }
    else
    {
      OtherState.insert(name);
    }

    if(religion == "Episcopalian")
    {
      Episcopalian.insert(name);
    }
    else if(religion == "Presbyterian")
    {
      Presbyterian.insert(name);
    }
    else if(religion == "Methodist")
    {
      Methodist.insert(name);
    }
    else
    {
      OtherReligion.insert(name);
    }

    if(age.substr(0, 1) == "4")
    {
      Forties.insert(name);
    }
    else if(age.substr(0, 1) == "5")
    {
      Fifties.insert(name);
    }
    else
    {
      Sixties.insert(name);
    }

    if(party == "(W)")
    {
      Whig.insert(name);
    }
    else if(party == "(D)")
    {
      Democrat.insert(name);
    }
    else if(party == "(R)")
    {
      Republican.insert(name);
    }
    else
    {
      OtherParty.insert(name);
    }

  }
  input.close();
}

int main()
{
  presidentNames();
  (Democrat + Whig).print();
  (VA & Episcopalian & Whig).print();
  (OH & Methodist).print();
  (Forties).print();
  (OH - Methodist).print();
  cout << ((Democrat & OH) <= (OH)) << endl;
  return 0;
}
```